Examining ASP.NET's Membership, Roles, and Profile - Part 17

By Scott Mitchell

Introduction

Many of the web applications I help build can be classified as in-production line of business applications that receive frequent and ongoing feature enhancements. Typically, these applications have dozens if not hundreds of users who rely on the site each and every day to accomplish tasks necessary to keep the company running smoothly. Every week or so the latest code is deployed to the production servers, bringing with it bug fixes and, very often, new features or changes to existing features. One challenge I've bumped into when working on such applications is how to best alert users of the new features and the changes to existing features?

One useful way to announce any important, system-wide changes is to do so immediately after the user signs into the site. A very simple way to accomplish this would be to automatically redirect all users to an announcements page after signing in, which would list information about the most recent updates to the application. However, this approach desensitizes the user to the announcement page since they are sent there each time they sign in, regardless of whether they've already seen the announcement. A better approach is to determine if there were any announcements made since the user last signed in. If so, redirect the user to a web page that displays just those unread announcements.

This article shows how to implement such a system using ASP.NET's Membership system, the Login control, and a dash of code. A complete working demo is available for download at the end of this article. Read on to learn more!

Creating a Database Table for Announcements

In order to show a user only those announcements that have been made since they last signed in, we need some way to store announcements and to associate a date with them. Perhaps the most flexible approach is to store the announcements in a database. If you download the demo available at the end of this article you'll find that the application contains a database named `ASPNETDB.mdf` in its `App_Code` folder. This database includes both the database objects used by the `SqlMembershipProvider` along with the `Announcements` table I created.

The `Announcements` table's schema (shown below) is fairly simple. Each announcement is uniquely identified by a uniqueidentifier column (`AnnouncementId`), and each announcement has a subject and body, modeled by the `Subject` and `Body` columns. The `DateCreated` field indicates the UTC date and time that the announcement was created, while the `Active` bit field provides a simple mechanism for announcements to be "turned off" without deleting them from the database.

Column	Data Type	Notes
`AnnouncementId`	`uniqueidentifier`	Primary key; default value of `NEWID()`
`Subject`	`nvarchar(50)`	
`Body`	`nvarchar(MAX)`	
`DateCreated`	`datetime`	Default value of `getutcdate()`
`Active`	`bit`	

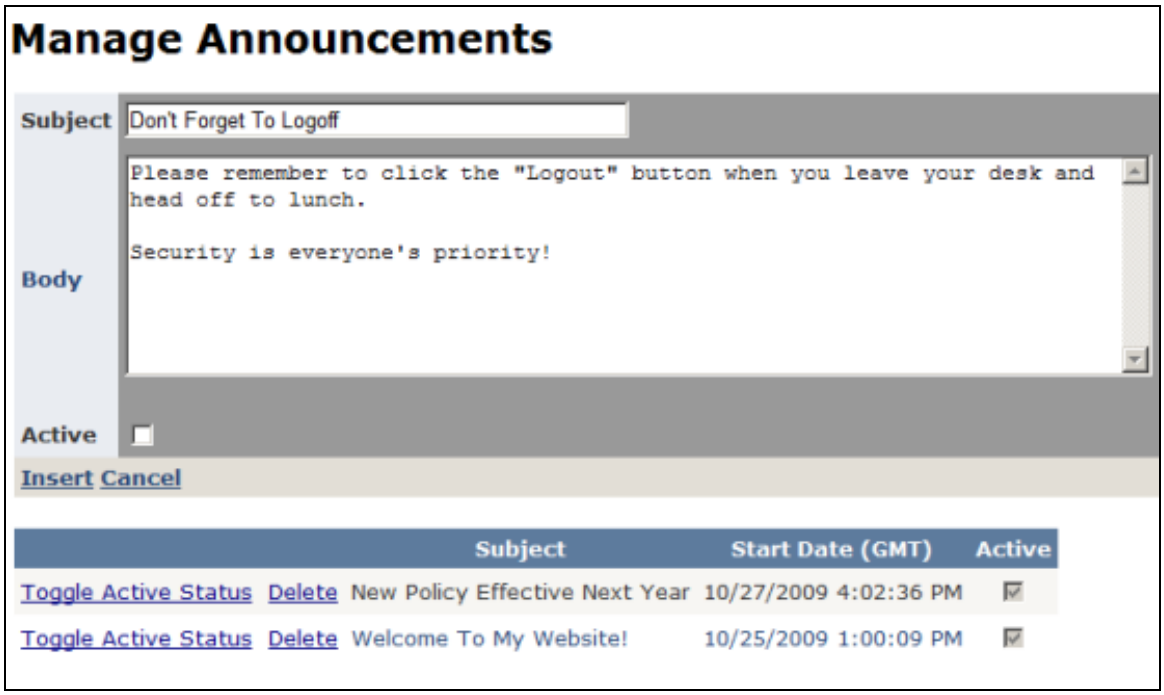
Anytime a new announcement is needed - whether there is a new rollout with new features to note or, perhaps, some business policy change that users must be made aware of - a new record should be added to the `Announcements` table. As we'll see later on in this article, when a user signs in he is shown all announcements that have been made since he last signed in. As a result, you don't need to worry about removing old announcements from the `Announcements` table. You can leave them in because users who have already seen them will never see them again.

Creating and Managing Announcements

The demo includes a simple ASP.NET page for creating and managing the records in the `Announcements` table (`~/UsersOnly/ManageAnnouncements.aspx`). This page contains a DetailsView, GridView, and SqlDataSource control that allow the user to:

- Add new announcements,
- Toggle the `Active` bit field of existing announcements, and
- Delete announcements.

The following screen shot show the `ManageAnnouncements.aspx` page in action.



Because the focus of this article is on displaying unread announcements to users when they sign on, I don't plan on exploring this page in detail. For a thorough look at using the GridView and DetailsView controls to display, insert, update, and delete data, refer to my article series Accessing and Updating Data in ASP.NET, along with my Data Access Tutorials.

Before we move on there are a couple of things I want to point out. First off, I have it so that the announcement's subject and body are plain text. When we examine the page that displays the unread announcements to the user who just signed on we'll see that the subject and body's content are HTML encoded, meaning that any HTML characters - such as < and > - are converted into their escaped equivalents - `<` and `>`. What that means is that if you type in HTML characters into the subject or body they will be literally displayed in the output.

Additionally, note that the `ManageAnnouncements.aspx` page is located in the `~/UsersOnly` folder. This folder uses URL authorization rules in the `Web.config` file in the `~/UsersOnly` folder to allow access to any authenticated user. Consequently, any user who can sign into the site can visit the `ManageAnnouncements.aspx` page and add and delete announcements. Chances are, you'd want to put this page in a folder that is locked

down to users in an administrative role. (Refer to Part 2 of this article series for an in-depth look at roles and role-based authorization, or check out my Website Security Tutorials.)

Determining Whether A User Has Unread Announcements

Whenever a user signs into the website we need to determine whether there are any unread announcements. To accomplish this we need to determine the date and time the user last signed in and compare that to the date and time of the most recent announcement. If the user's last sign in time predates the most recent announcement then there exists at least one unread announcement. However, if the user's last sign in time is later than the most recent announcement then there are no unread announcements.

The Membership system automatically tracks users' last login date/times, specifically via the `MembershipUser` class's `LastLoginDate` property. Keep in mind that whenever a user is authenticated the Membership system automatically updates this value to the current UTC date and time. Therefore, it's imperative that we capture this value before the user is actually authenticated so that we get the date and time they logged in prior to their current log in.

If you are using the Login Web control to authenticate users then you will need to create two event handlers:

- One for the `LoggingIn` event, where we will get and store the `LastLoginDate` value for the user signing into the site, and
- One for the `LoggedIn` event, where we will determine the date of the most recent announcement and compare that to the user's last login date, redirecting the user to a page to view the announcements, if necessary.

As I just noted, after the user is authenticated the `LastLoginDate` value reflects the current UTC date and time. That is the reason why we will need to create an event handler for the Login control's `LoggingIn`. The `LoggingIn` event fires before the user is authenticated and gives us an opportunity to get the date and time of their last login. If we waited until the `LoggedIn` event, which fires after the user is authenticated, the `LastLoginDate` value would be the current UTC date and time and not the date and time the user had previously signed in.

The following code shows the `LoggingIn` and `LoggedIn` event handlers. I've removed the data access code from the `LoggedIn` event handler for brevity.

```
Dim usersLastLoginDate As DateTime = DateTime.MaxValue

Protected Sub myLogin_LoggingIn(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.LoginCancelEventArgs) Handles myLogin.LoggingIn
   'Get info about the user
   Dim usrInfo As MembershipUser = Membership.GetUser(myLogin.UserName)

   If usrInfo IsNot Nothing Then
      'This user account exists... "save" their last login date so that it can be examined in the
LoggedIn event handler
      usersLastLoginDate = usrInfo.LastLoginDate.ToUniversalTime()
   End If
End Sub

Protected Sub myLogin_LoggedIn(ByVal sender As Object, ByVal e As System.EventArgs) Handles
myLogin.LoggedIn
   ... Some data access code removed for brevity ...


   'Determine the most recent announcement date
   Dim mostRecentAnnouncementDate As DateTime = ... Get Most Recent Announcement Date ...

   'See if this user's last login date precedes the most recent announcement
   If usersLastLoginDate < mostRecentAnnouncementDate Then
```

```
        Response.Redirect(String.Format("~/Announcements.aspx?AnnouncementsSince={0}&RedirectUrl=
{1}", _
                                  Server.UrlEncode(usersLastLoginDate), _
                                  Server.UrlEncode(FormsAuthentication.GetRedirectUrl(myLogin.UserName,
False))))
    End If
End Sub
```

Note that the `usersLastLoginDate` variable is defined outside of the two event handlers, meaning that both event handlers have access to this variable. The `LoggingIn` event handler fires first, after the user enters their username and password and clicks the "Log In" button. This event handler starts by using the `Membership.GetUser(`*username*`)` method to retrieve information about the user signing into the site. Of course, at this point we don't know if the user credentials supplied are valid. The person signing into the site may have entered a username that does not exist in the database. For that reason, it's vital that we ensure that the `Membership.GetUser(`*username*`)` method actually returned a `MembershipUser` object before we attempt to read the `LastLoginDate` value. Assuming there is a user account with the specified name, we store the UTC value of the `LastLoginDate` property to the `usersLastLoginDate` variable.

The `LoggedIn` event fires after the `LoggingIn` event and only fires if the supplied credentials are valid. In this event handler we start by getting the UTC date and time of the most recent announcement. (I've removed this data access code from the above code snippet for brevity, but we'll come back to it shortly.) This value is then compared against the user's last login date value retrieved from the `LoggingIn` event handler. If the user's last login date precedes the most recent announcement date then we redirect the user to `Announcements.aspx`, which will show the reader her unread announcements.

Note that in the redirect to `Announcements.aspx` we are passing along two querystring values:

- `AnnouncementsSince` - the UTC date/time value of when the user last logged into the site, which is used by `Announcements.aspx` to determine which announcements need to be displayed, and
- `RedirectUrl` - the URL the login page was going to send the user to after logging in. This value is passed to the `Announcements.aspx` page so that after the user reads the announcements she can continue onto the page she would have been sent to had there been no unread announcements. The value for this parameter comes from the `FormsAuthentication.GetRedirectUrl` method.

Also note that the parameters injected into the querystring are appropriately encoded via the `Server.UrlEcode` method. Whenever you dump a variable into the querystring you should always be sure to `UrlEncode` it for reasons discussed in this article: Using `Server.UrlEncode`.

Determining the Date and Time of the Most Recent Announcement

In addition to adding the `Announcements` table to the `ASPNETDB.MDF` database, I also created a stored procedure named `MostRecentAnnouncementDate` that returns the date and time of the most recent announcement (more specifically it returns the date and time of the most recent announcement whose `DateCreated` value is less than or equal to the current UTC date and time). The query used by the `MostRecentAnnouncementDate` stored procedure follows:

```
SELECT MAX(DateCreated)
FROM Announcements
WHERE DateCreated <= getutcdate()
```

The `LoggedIn` event handler code in the demo uses ADO.NET to connect to the database and execute the `MostRecentAnnouncementDate`.

Displaying Unread Announcements

The final piece of the puzzle is the `Announcements.aspx` page, which displays the unread announcements. Recall that the user reaches this page from the login page (`~/Login.aspx`) if there are more recent

announcements since they last logged on. Moreover, two bits of information are passed to the `Announcements.aspx` via the querystring: the UTC date and time the user last logged on (`AnnouncementsSince`) and the URL the user would have been sent to had there been no unread announcements (`RedirectUrl`).

The `Announcements.aspx` page contains a Repeat Web control that's bound to a SqlDataSource control. The SqlDataSource control executes the following ad-hoc query:

```
SELECT [Subject], [Body]
FROM [Announcements]
WHERE (([Active] = @Active) AND
      ([DateCreated] >= @DateCreated) AND
      ([DateCreated] <= getutcdate()))
ORDER BY [DateCreated]
```

The `@Active` parameter's value is set to `True` in the SqlDataSource control's `SelectParameters`, thereby only retrieving active announcements. The `@DateCreated` parameter's value is assigned to the `AnnouncementsSince` querystring value. That conditional, in conjunction with `[DateCreated] <= getutcdate()`, returns those announcements that have occurred after the user last logged on but before the current date and time. In total, this query returns all active announcements between the user's last login date and the current date/time, ordered so that the oldest announcements are displayed first.

The Repeater control displays these results using a series of `<div>` elements. Here's the markup specified in the Repeater's `ItemTemplate`:

```
<div class="announcementSubject">
   <%#Server.HtmlEncode(Eval("Subject").ToString())%>
</div>
<div class="announcementBody">
   <%#FormatAnnouncementBody(Eval("Body").ToString())%>
</div>
```

Note that the `Subject` field's output is HTML encoded via the `Server.HtmlEncode` method. This ensures any markup entered into the subject line for the announcement is appropriately escaped. The `Body` field is not displayed directly, but rather passed into the `FormatAnnouncementBody` formatting function, which is defined in the ASP.NET page's code-behind class:

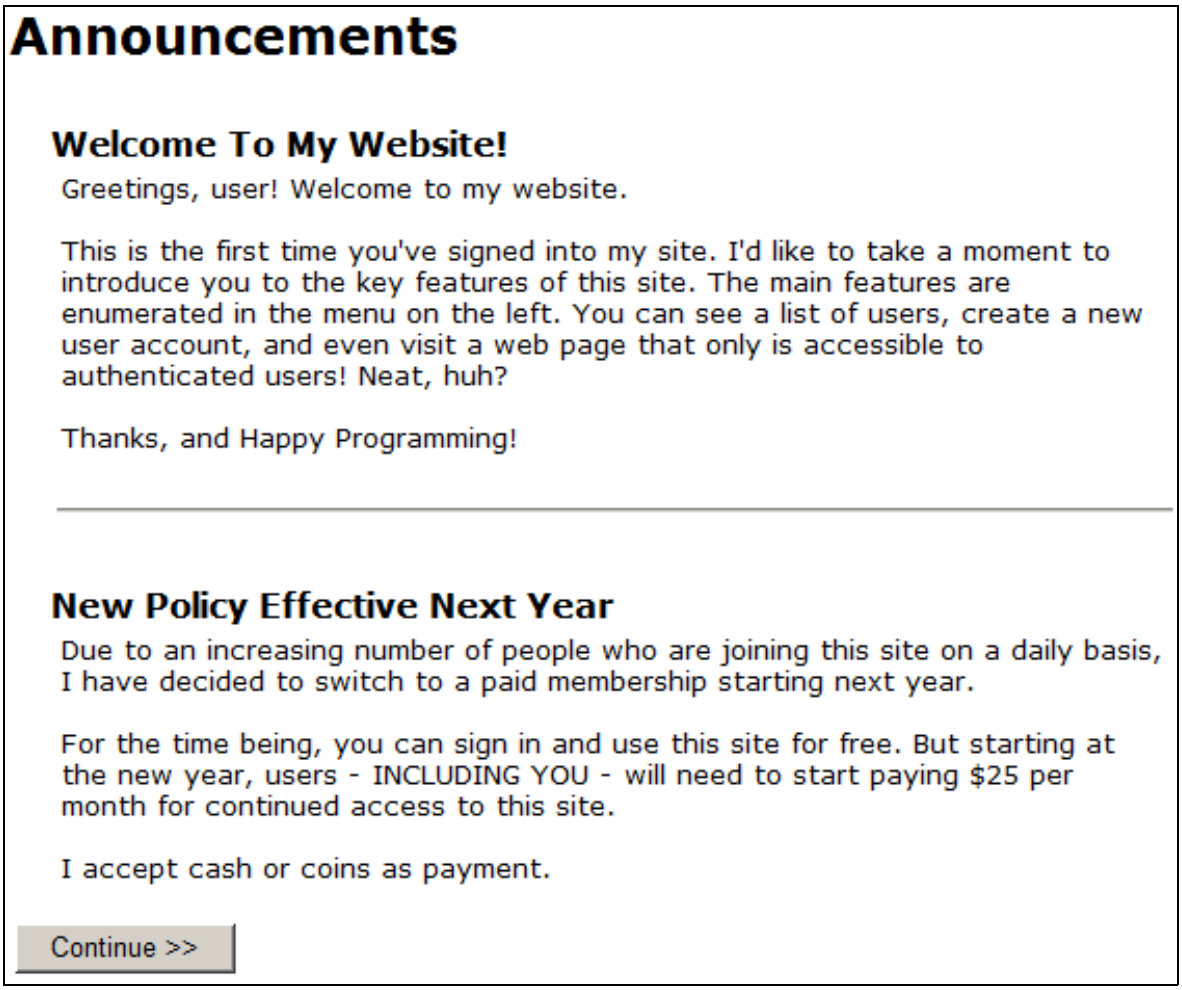
```
Protected Function FormatAnnouncementBody(ByVal body As String) As String
   Dim output As String = Server.HtmlEncode(body)

   output = output.Replace(Environment.NewLine, "<br />")

   Return output
End Function
```

As you can see, this method takes the body value, HTML encodes it, and replaces carriage returns with a `<br />` tag.

An End-To-End Example...

With all of these pieces in place, let's look at the user experience. Imagine that we have created two announcements, one on October 25th, 2009 at 1:00 PM GMT and the other on October 27th, 2009 at 4:02 PM GMT. Assume that a user signs into our site on October 28th, 2009 3:15 PM GMT, and that the time he signed on before that was on October 21st 2009 9:45 AM GMT.

Because the user's last login date and time (October 21st 2009 9:45 AM GMT) precedes the most recent announcement date and time (October 27th, 2009 at 4:02 PM GMT) the user is automatically redirected from the login page to the `Announcements.aspx` page, where he is shown both announcements (because both announcements were created after his previous login date/time).



If the user had last logged in on October 26th 2009 11:45 AM GMT, he would still be redirected to the `Announcements.aspx` page because his last login date/time precedes the most recent announce date and time (October 27th, 2009 at 4:02 PM GMT), but the `Announcements.aspx` page would show only one announcement. The older announcement - the one from October 25th, 2009 at 1:00 PM GMT - would not be displayed because it was created prior to the user's last login date time. Presumably, the user saw this announcement when he last logged in on the 25th.

As the screen shot above shows, the `Announcements.aspx` page also contains a Continue button. This Button's `Click` event handler does a `Response.Redirect` to the URL specified via the `RedirectUrl` querystring parameter, which sends the user off to the page they would have originally been directed to had there been no new announcements for them to read.

Ideas for Enhancements

The code and concepts put forth in this article provide a robust, easy to use announcement system. In my projects I've implemented a number of bells and whistles that are not shown here, including functionality like:

- **Rich announcement messages** - the use of a rich textbox and the ability to include HTML in the announcement body. This means that announcements can have formatted text, including images and hyperlinks.
- **Role-based announcements** - currently, all announcements are shown to all users. However, certain new features or important messages might be targeted to specific classes of users. With a bit of work you could augment this system so that when creating an announcement you can (optionally) specify that the announcement applies only to a certain subset of roles. Then, only users in those roles would see those announcements.
- **Adding "I have read this announcement" CheckBoxes** - one site I worked on used announcements to

display important legal messages to its end users. It was important that they had proof that their users had read these announcements and didn't somehow skip over them or simply click "Continue" without reading them. To this end we enhanced the Repeater in the `Announcements.aspx` page to include a CheckBox titled "I have read this announcement" along with each announcement. The user could not continue until each announcement's checkbox was checked. Moreover, we'd record in the database the date and time the user checked this checkbox.

Happy Programming!

- By Scott Mitchell

Further Reading

- Accessing and Updating Data in ASP.NET
- Data Access Tutorials (VB and C# versions available)
- Website Security Tutorials (VB and C# versions available)

Attachments

- Download the code used in this article

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 17
Article Author:	Scott Mitchell
Published Date:	November 4, 2009
Article URL:	http://www.4GuysFromRolla.com/articles/110409-1.aspx